3. What is a Buffer Overflow Attack?

A **Buffer Overflow Attack** is an exploit where an attacker deliberately sends more data than a buffer can handle to overwrite memory adjacent to the buffer. By doing so, the attacker can manipulate the program to execute malicious code, crash, or behave unpredictably.

How a Buffer Overflow Attack Works:

1. **Locate a Vulnerable Program**:
   The attacker finds a program that uses unsafe functions (e.g., `strcpy` or `gets`) without proper input validation.
2. **Overflow the Buffer**:
   They send input larger than the buffer size, causing the extra data to overwrite adjacent memory.
3. **Manipulate Memory**:
   The attacker crafts the input to overwrite specific areas, such as:
   - **Return Address**: Redirecting program execution to malicious code (e.g., a payload).
   - **Variables**: Changing program behavior.
4. **Execute Malicious Code**:
   By controlling the program's flow, the attacker can execute shellcode, open a backdoor, or escalate privileges.

Anatomy of an Attack:

Here's an example of a **stack-based buffer overflow attack**:

Vulnerable Code:

```c
#include <stdio.h>
#include <string.h>

void vulnerable_function(char *input) {
    char buffer[10];  // Small buffer with 10 bytes
    strcpy(buffer, input);  // No size check: unsafe!
    printf("You entered: %s\n", buffer);
}

int main() {
    char input[100];
    printf("Enter input: ");
    gets(input);  // Unsafe: no size check
    vulnerable_function(input);
```

```
    return 0;
}
```

Exploit Input:

Suppose the attacker enters this carefully crafted string:

```
AAAAAAAAAA<malicious_code_address>
```

- The first `AAAAAAAAAA` fills the buffer.
- The extra data overwrites the **return address** with the address of the attacker's code.
- When the function returns, it jumps to the attacker's code instead of the original return address.

Real-Life Impact:

1. **Code Execution**: Attackers can execute custom payloads to steal data, install malware, or open backdoors.
2. **Privilege Escalation**: Gaining unauthorized administrative rights.
3. **Denial of Service (DoS)**: Crashing the application or server.

Famous Buffer Overflow Exploits:

1. **Morris Worm (1988)**: The first major internet worm used buffer overflow to spread.
2. **Blaster Worm (2003)**: Exploited a buffer overflow in Windows DCOM RPC.
3. **Heartbleed Bug (2014)**: Though not a classic overflow, it abused memory allocation to leak sensitive data.

Tools Used in Buffer Overflow Attacks:

1. **GDB (GNU Debugger)**: For analyzing and exploiting memory.
2. **Immunity Debugger**: For debugging Windows programs.
3. **Metasploit Framework**: To craft and execute payloads.
4. **Fuzzer Tools**: To find vulnerabilities in programs.

How to Prevent Buffer Overflow Attacks:

1. **Input Validation**: Always validate the size and type of input.
2. **Safe Programming Practices**: Use functions like `fgets` instead of `gets`, and avoid `strcpy` or `sprintf`.
3. **Compiler Protections**:
    - **Stack Canaries**: Insert a value in the stack to detect overwrites.
    - **ASLR (Address Space Layout Randomization)**: Makes it hard to predict memory layout.

4. **Modern Languages**: Use languages like Python or Java that manage memory safely.